SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: NRL Report 7906
2. GOVT ACCESSION NO.:
3. RECIPIENT'S CATALOG NUMBER:

4. TITLE *(and Subtitle)*: FILE PARTITIONING AND RECORD PLACEMENT IN ATTRIBUTE-BASED FILE ORGANIZATIONS

5. TYPE OF REPORT & PERIOD COVERED: An interim report on a continuing NRL problem.

6. PERFORMING ORG. REPORT NUMBER:

7. AUTHOR(s): Edwin J. McCauley, Frank A. Manola

8. CONTRACT OR GRANT NUMBER(s):

9. PERFORMING ORGANIZATION NAME AND ADDRESS: Naval Research Laboratory, Washington, D.C. 20375

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: NRL Problem B02-24; Project RF-21-222-401-4361

11. CONTROLLING OFFICE NAME AND ADDRESS: Department of the Navy, Office of Naval Research, Arlington, Virginia 22217

12. REPORT DATE: July 10,1975

13. NUMBER OF PAGES: 13

14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)*:

15. SECURITY CLASS. *(of this report)*: Unclassified

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE:

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

Attributes	Data Base Task Group	File structures	Physical structure
Clusters	Data structures	Keywords	Record access
CODASYL	DBMS	Logical structure	Record placement
Data bases	File partitioning	Modeling	Storage cells

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

The position occupied by a record of the data base on secondary storage can affect performance in a variety of ways. Record placement and file organization interact with one another. A model was developed for certain techniques that utilize the physical device characteristics and the logical file content to optimize retrieval efficiency and precision. Methods were reviewed for partitioning the file into disjoint groups of records (called clusters) such that in most cases an access to the file will involve a small number of clusters. A record placement technique that preserves these clusters was developed, followed by a search algorithm which, when this record

(Continued)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

placement policy is followed, gives improved performance in the areas of precision and efficiency. Finally, application of the techniques to existing systems was considered.

CONTENTS

FILE PARTITIONING AND RECORD PLACEMENT IN ATTRIBUTE-BASED FILE ORGANIZATIONS

INTRODUCTION

The design of file structure for a data base management system (DBMS) requires that many different features be taken into account. The designer must be simultaneously cognizant of the physical characteristics of the devices on which the file is to be located, of the logical content of the data base, and of anticipated uses of the system. File designers have always been concerned about record placement and file organization. Where a record of the data base is placed on secondary storage can affect performance in a variety of ways. For example, the first access to certain contiguous units of secondary storage "cells" (e.g., a cylinder on a moving-arm disk or a page in a virtual memory system) "costs" more than immediately subsequent accesses to the same cell, since a price is paid for the first access (arm movement for the cylinder, a transfer to main memory for the page) which is not paid for the subsequent accesses. At the same time, when the DBMS is given a query requesting some set of records from the data base, some file organizations will allow that set of records to be precisely identified in the logical file structure without accessing numerous records unrelated to the query, while other file organizations will not. Moreover, record placement and file organization interact with one another. For example, in the case described above, optimum performance would be provided if the set of records could be precisely located in the logical file structure, and if the set had been placed in the fewest possible storage cells. In this way the fewest possible "first access costs" would be paid. Of course, it may not be possible to provide such optimal placement for every anticipated query unless many duplicate records are allowed.

A model has been developed for certain techniques which utilize the physical device characteristics and the logical file content to optimize retrieval efficiency and precision. Methods are suggested for partitioning the file into disjoint groups of records (called clusters) such that in most cases an access to the file will involve a small number of clusters. A record placement technique which preserves these clusters is developed. Then a search algorithm is developed which, when this record placement policy is followed, gives improved performance in the areas of precision and efficiency. Finally, application of the techniques to existing systems is considered. This work is in many respects an extension of work reported in Refs. 1-6.

DATA BASE MODEL

We start with two undefined terms: a set A of attributes and a set V of values. These are left undefined to allow the broadest possible interpretation.

Note: Manuscript submitted May 1, 1975.

A record r is a subset of the Cartesian product A X V in which each attribute has one and only one value. We can consider r to be a collection of ordered pairs: (an attribute, its value). Every record is assigned a unique address.

For practical reasons we shall limit our consideration to (attribute, value) pairs which are succinct. Such pairs will be called keywords, and will be denoted by K or Ki. Although larger pairs will still be allowed to occur in records and will be returned when the record is retrieved, only keywords may be used in queries.

Relations on Records

A relation on the records of a data base is a set of ordered pairs of records (r, r'). If two records are related by a relation REL, we denote this by r REL r'. REL is an *equivalence* relation if it has the following properties:

- Reflexive: for any record r, r REL r
- Symmetric: if r REL r', then r' REL r
- Transitive: if r REL r' and r' REL r'', then r REL r''.

An equivalence relation partitions the records of the data base into disjoint groups called equivalence classes. For a given equivalence relation REL, only records within a single equivalence class are related to one another by REL. Two records in different equivalence classes of REL are not related by REL. (They may be related in some other way, however.)

We shall now identify a number of useful equivalence relations on the records of a data base. The first relation is derived from physical considerations.

Cell Equivalence

Two addresses a and a' are related by CEA, cell equivalence, if

$$a = e * s + d$$

and

$$a' = e * s + d'$$

where $0 \leqslant d < s$ and $0 \leqslant d' < s$. The partitions introduced by CEA will be called secondary storage cells, or simply cells. The number of cells is one greater than the maximum value of e. The integer s is called the cell size. Since this relation is defined for every address, an address *a* can be associated with a cell as follows:

c:=ENTIER (a/s), ENTIER is the ALGOL "floor function."

The cell equivalence relation on addresses, CEA, determines an equivalence relation on records, CER, based on the physical placement of the records. Two records are related by CER if their addresses are related by CEA, i.e., if the two addresses are in the same cell.

Logical Equivalence

Another category of equivalence relations is based on the logical content of the record as revealed by its keywords. In earlier work, both Wong and Chiang [6] and Rothnie and Lozano [5] have discussed such logical equivalence relations.

A keyword K is true for a record r if K is in r; otherwise, it is false. A query is a proposition given by a Boolean expression of keywords. A query is true for a record r if this proposition holds for the keywords of r. In this case, r is said to satisfy the query. Thus, every Boolean expression f(K1, ..., Km) is either true or false for each record.

Consider a set of keywords X = [K1, ..., Kn]. We shall base our logical equivalence relation upon expressions of the following form;

$$K1^* \wedge K2^* \wedge \ldots \wedge Kn^*$$

where Ki* is either Ki or $\overline{Ki}$. There are 2**n such expressions. A given record will satisfy one and only one of these expressions. Therefore, it should be clear that these expressions each determine an equivalence class in the logical equivalence relation LOG. In practice, it is likely that many of these equivalence classes are empty because no records satisfy that expression of the keywords in X.

Structural Equivalence

A third type of equivalence relation is structural. The structural equivalence relation STR models those partitioning decisions which are based on factors other than the record's address or its keywords. The most important of these other types of partitioning result from design decisions about the file structure. For example, in an inverted file the designer, in effect, specifies that no record is related to any other record by the file structure, i.e., that the equivalence classes of STR have exactly one member each. It is also desirable to limit the size for the STR equivalence classes for reasons other than to produce an inverted file, such as to limit the propagation of damage. If, for example, all the records of the file were on a single chain, i.e., there is one STR equivalence class with all the records in it, then damage to any record might render unreachable all those records further down the chain. We emphasize the separation between STR and CER because STR models the results of design decisions and CER models the result of physical device characteristics.

Cluster Equivalence

The three equivalence relations already discussed, CER, LOG, and STR, may be intersected to form a new equivalence relation. This equivalence relation, which we shall call cluster equivalence, (CLS), has the property that two records r and r′ are related by CLS,

i.e., r CLS r′, if and only if r CER r′, r LOG r′, and r STR r′. We shall call the equivalence classes of CLS clusters.

The use of the concept of cluster equivalence allows us to simultaneously reflect desirable logical, physical, and structural properties. We will see that an insertion algorithm which preserves these logical, physical, and structural properties allows us to base an access algorithm on the cluster equivalence relation with assurance that efficient access will be performed.

When a system user presents a record to be inserted in the data base, the keywords of the record determine the logical equivalence class to which the record belongs by determining which of the expressions of the keywords from X the record satisfies. The system decides where the record is to be placed, which determines to which cell equivalence class the record will belong. The record's membership in a structural equivalence class is determined by how the system links the record into the data base structure.

File Structure and Algorithms

To discuss this, we must add to our basic model the data base structures and algorithms needed for record insertion and retrieval.

Associated with each keyword K in record r is the address of another record with the same keyword. We shall call this the pointer of r with respect to K or briefly the K-pointer. To retain the uniformity of definitions, we allow the existence of null pointers.

A list L of records with respect to keyword K (or, briefly, a K-list) is a set of records each containing K such that

1. The K-pointers are all distinct,
2. Each non-null K-pointer gives the address of a record within L and L only,
3. There is a unique record in L not pointed to by any other record containing K, called the beginning of the list,
4. There is a unique record in L with a null K-pointer, the end of the list,
5. No two Ki lists have a record in common for the same i.

We let p[i,j] be the number of records in the jth Ki list, which has beginning address a[i,j].

A set F of records is called a file if every K-list containing one or more of these records is contained in F. Every file is assigned a unique file name.

A directory of F is a set of sequences, one for each keyword Ki, called directory entries for Ki and denoted D(Ki), each of which is of the form

D(Ki) = [Ki, h[i]; (c[i,1], a[i,1], p[i,1], ...,

(c[i,h[i]], a[i,h[i]], p[i,h[i]])],

where

h[i] is the number of clusters which include records containing Ki

c[i,j] is the cluster in which the jth Ki list is contained

a[i,j] and p[i,j] are as defined above.

We shall require that every K-list have all its member records in the same cluster. We shall call this the disjoint list property. This property, together with the fact that there is an entry in the directory for each distinct K-list which identifies the cluster in which that list is contained, allows clusters to be excluded from possible search through examination of only the directory if the cluster cannot contain records which satisfy a query.

Record Insertion

Algorithm 1 — Record Insertion:

Input: A record to be inserted.

Step 1: Determine which of the keywords of the record are in the set X. These keywords will determine the logical equivalence class to which the record belongs.

Step 2: Determine candidate clusters for insertion by intersecting the sets of cluster identifiers associated with each of the keywords found in Step 1. Call the resulting set of cluster identifiers THETA.

Step 3: Delete from THETA those clusters into which the record cannot be inserted either because the cell in which the cluster is located is full or because insertion in the cluster is not allowed by the structural relation.

Step 4: Examine the remaining clusters represented in THETA against the added record, deleting any cluster which represents a different logical equivalence class.

Step 5: If THETA is null, add to THETA the address of the location where a new cluster may be started. This new cluster location should be selected so that all the members of the logical equivalence class are in as few cells as possible.

Step 6: Arbitrarily select any remaining member of THETA, insert the record in that cluster, link it onto all the appropriate Ki-lists, and update the directory.

Notice that this algorithm does not make use of any secondary data structures other than the directory and the temporary structure THETA. The algorithm preserves the logical

relation (step 4) and the structural relation (step 3). Steps 2 and 3 eliminate from consideration at an early stage those clusters into which the record could not be inserted.

Record Retrieval

Algorithm 2 — Parallel Search:

Input: A query as an expression of keywords in disjunctive normal form.

Step 1: Determine candidate clusters by intersecting the set of cluster identifiers associated with the keywords in the ith conjunct, placing the results in THETA[i]. Repeat for each of the conjuncts in the query.

Step 2: Find the prime keywords. Find the keyword KPRIME in the ith conjunct that has the shortest KPRIME-list in each cluster in THETA[i]. Merge a pair (KPRIME, c) onto a list GAMMA where c is the cluster in which KPRIME is a prime keyword. Repeat for each of the conjuncts in the query.

Step 3: Decode the directory and initialize the search list. For each (KPRIME, c) pair on GAMMA, find the beginning address of the KPRIME-list in cluster c and merge this address onto an ascending sorted list, SIGMA.

Step 4: Search the file.

a. Find the smallest unused address on SIGMA and retrieve that record. Mark that address as used. If all addresses have been used, exit.

b. For each of the keywords K that are prime for this cluster and that are present in the retrieved record, merge their K-pointers into SIGMA.

c. Process the retrieved record against the user's query. If the record satisfies the query, give the record to the user.

d. Continue with step 4a.

This retrieval algorithm has several desirable characteristics. First, some clusters will be immediately excluded from consideration in step 1 because they cannot contain records which satisfy the query. If the structural equivalence classes are defined so as to give an inverted file, this same step gives the 100% precision characteristic of inverted file processing. By selecting prime keywords we examine the minimum number of records. This is because in searching down a prime keyword list the algorithm will encounter all the records in the cluster which could possibly satisfy the query. Notice that a different keyword from a conjunct may be prime for different clusters. The algorithm retrieves a given record once, even if that record is on the lists of many keywords that are prime for this cluster. The literature [3,4,7] has examples of how keyword lists are searched in parallel by a similar algorithm. Because of the disjoint list property discussed above, the processing of records in one cluster can be made independent of that in any other cluster. Thus it may

be possible to process different clusters in parallel if the physical input-output (I/O) system allows for such operations.

CONCLUSIONS

This approach partitions the set of records in a file into clusters, such that all the records of a given cluster have certain logical characteristics in common. Given our assumptions concerning the costs of accessing cells, a reasonable objective of a search algorithm is to minimize the number of cells accessed in retrieving records satisfying some Boolean expression of keywords. To attain this objective, two conditions must be satisfied:

- Records must be assigned to cells in such a way that a small number of cells will contain records satisfying the Boolean expression.

- Some mechanism must exist for determining which cells contain these records without accessing other cells.

Since all the members of a cluster are stored in a minimum number of cells, and information about this mapping is maintained in the directory, both these conditions are fulfilled. Because a cluster is a logical, rather than a physical, partition, the number of records in a cluster will vary depending on the distribution of values in the file. Hence, the cluster may contain more records than will fit on a single cell. Nevertheless, as long as care is taken to minimize the number of cells containing records of a given cluster, the conditions are still satisfied.

Let us reflect briefly on the implications of these results. First, note that a query made up only of keywords from X results in the retrieval of only records satisfying the query, i.e., 100% retrieval precision. Thus, if X is chosen to include the most frequently used keywords, we will have a high probability that an arbitrary query will be answered with 100% retrieval precision. Such a situation can be, of course, achieved by using an inverted file, but this model allows for a substantially smaller directory while achieving nearly the same retrieval precision as an inverted file. The update algorithm places all the records from a logical equivalence class in as few cells as possible. The search algorithm examines only those clusters which could contain records which satisfy the query. So, for each conjunct, we will access a minimum number of cells and return all the records which satisfy the conjunct and only those records.

OTHER APPLICATIONS

The preceding results can also be applied to other situations. Consider the problem of partitioning the data base into security classes as discussed in Ref. 8. We would like to avoid the necessity of reorganizing the data base every time the access rules are changed. At the same time we must have no more than one security class in a cell, so that physical compartmentalization may be used to reinforce the logical access controls. This model provides a solution to both of these problems. If X includes all keywords which may be used in formulating an access rule, every record in a cluster will have the same set of

accesses permitted and denied regardless of what access rules are actually in force. This is because within a cluster all the records satisfy the same Boolean expression of the keywords in X. Since X contains all the keywords from which access rules may be formulated, any access rule simply permits or denies access to an entire cluster.

The use of these concepts is not merely confined to theory. The motivations for the model and the approach described in this paper closely parallel certain aspects of the work of the CODASYL Data Base Task Group [9] and those of its successor, the Data Description Language Committee [10]. In the following discussion, some acquaintance is presumed with these specifications.

We will consider the relationship of our model to the DBTG specifications by considering a particular mapping of language constructs from Refs. 1 and 2 to the concepts of our model. Simplifying considerably, the DBTG "set" construct, which links related records logically, becomes a list (or lists) in our model. The DBTG area, which partitions the records of the data base, becomes a cluster. An area is defined in Ref. 10 as:

> An AREA is a named collection of records which need not preserve [set] relationships. An area may contain occurrences of one or more record types, and a record type may have occurrences in more than one area. A particular record is assigned to a single area and may not migrate between areas.

The following discussion from Ref. 10 of some of the uses of sets and areas clarifies the relationship between the DBTG concepts and those of our model:

> The concept of area allows the Data Administrator to subdivide a data base rather than considering the data base as a single unit. The use of areas allows the Data Administrator or the DBMS to control placement of an entire area to provide efficient storage and retrieval .... The objective of providing for control of relative placement of records is to increase efficiency by advising the DBMS of anticipated usage patterns of records. Thus, the schema DDL permits specification of the area or areas to which occurrences of a particular record type are to be assigned by the DBMS. The schema DDL also includes a clause which causes records being added to the data base to be stored near some other record in the data base. Conceptually, the effect of such clauses is to request the clustering of records which are required as a group to perform some procedure, thereby improving performance.

Those familiar with the DBTG specifications will note immediately that DBTG sets do not necessarily satisfy the same disjoint list property with respect to areas as lists do with respect to clusters in our model, since, in general, a set may contain records residing in several areas. If a set contains records in different areas, and if we assume a simple mapping from different areas to different physical files (as is often the case in DBTG implementations), access via the logical structure (sets) of the data base may be nonlocal in its access to physical storage cells which have been associated with the files, thus creating inefficient physical access characteristics. For this reason, the DBTG specifications

include the capability of making sets satisfy a disjointness property with respect to areas similar to our disjoint list property, if retrieval via these sets is to be optimized. This is done through a combination of record-to-set and record-to-area assignment policy, which ensures that records in a given set are confined to the same area (and under our assumed physical storage mapping, to a small set of physical storage cells).

Of course, one must not attempt to carry too far this analogy between the specific details of our model and those of the DBTG specifications. For example, the area is not precisely a cluster because the physical units of storage to which the area is mapped do not partition the sets of an area, and thus the sets would not satisfy the disjoint list property with respect to cells, as they must in our model. Further, the area construct can be used for purposes other than logical clustering, such as the grouping of records for recovery purposes.

What we are trying to do in discussing these DBTG concepts is to show that the concepts discussed in our model have been taken into account in some modern data base systems. We are not necessarily claiming that the DBTG systems will exhibit all the properties of this model, since some properties in particular depend upon the specific DBTG implementation. Furthermore, it is by no means true that only DBTG implementations can use the principles presented here. The model is not concerned with the user view of the data base, but rather with internal techniques. Thus, the principles described here can also be applied to systems based on the relational model of data introduced by Codd [11]. For example, Whitney [12] describes several systems based on the relational model which either could use, or presently do use, techniques similar to those described here. Also, Rothnie has built a relational-model-based DBMS described in Refs. 13 and 14 that uses the model he describes in Ref. 5, which we have indicated is closely related to the model described here.

SUMMARY

We have presented a formal model for file structures and algorithms for the use of these structures. Systems which are constructed to use the ideas discussed above will likely have a high retrieval precision and a minimum number of cell accesses for a larger number of queries.

A system based directly on this model is currently in the design stage at Ohio State University. We hope to report on concrete results in the near future.

ACKNOWLEDGMENTS

We would like to thank our good friend and mentor, Dr. David Hsiao of Ohio State University for providing the stimulus for this report. Without the efforts of our reviewers, Dr. Harvey Koch of Ohio State University and A. Metaxides of Bell Laboratories, this report would have been far less complete.

REFERENCES

1. D. Hsiao and F. Harary, "A Formal System for Information Retrieval from Files," Comm. ACM 13, No. 2, 67-73 (Feb. 1970).

2. D. Hsiao and F. Manola, "Data Management with Variable Structure and Rapid Access," *Proceedings of the First USA-JAPAN Computer Conference*, Tokyo, Japan, 1972.

3. D. Hsiao and F. Manola, "A Unified Approach to Structure, Access and Update in Data Base Systems," *Proceedings of EUROCOMP 1974*, May 1974.

4. F. Manola and D. Hsiao, "A Model for Keyword Based File Structure and Access," NRL Memorandum Report 2544, Naval Research Laboratory, Washington, D.C., Jan. 1973.

5. J.B. Rothnie, Jr., and T. Lozano, "Attribute Based File Organization in a Paged Memory Environment," Comm. ACM 17, No. 2, 63-69 (Feb. 1974).

6. E. Wong and T.C. Chiang, "Canonical Structure in Attribute Based File Organization," Comm. ACM 14, No. 9, 593-597 (Sept. 1971).

7. D. Hsiao, *System Programming - Concepts of Operating and Data Base Systems*, Addison-Wesley, to be published 1975.

8. E. McCauley, "A Model for Data Secure Systems," Ph.D. Dissertation, Ohio State University, 1975.

9. *CODASYL Data Base Task Group, April 1971 Report*, Association for Computing Machinery, 1971.

10. *CODASYL Data Description Language Journal of Development*, NBS Handbook 113, U. S. Government Printing Office, Washington, D.C., June 1973.

11. E.F. Codd, "A Relational Model of Data for Large Shared Data Banks," Comm. ACM 13, No. 6, 377-387 (June 1970).

12. V.K.M. Whitney, "Relational Data Management Implementation Techniques," *Proceedings of the 1974 ACM SIGFIDET Workshop*, ACM, New York.

13. J.B. Rothnie, "The Design of Generalized Data Management Systems," Ph.D. Dissertation, Dept. of Civil Engineering, MIT, 1972.

14. J.B. Rothnie, "An Approach to Implementing a Relational Data Management System," *Proceedings of the 1974 ACM SIGFIDET Workshop*, ACM, New YORK.